МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №1

по дисциплине «Алгоритмы и структуры данных»

Тема: Рекурсия

Студент гр. 8304 _____ Карабанов Р.Е

Преподаватель _____ Фирсов М.А

Санкт-Петербург

2019

Цель работы.

Ознакомиться с приёмами рекурсивного программирования, получить навыки программирования рекурсивных функций.

Задание.

Вариант № 24.

Построить синтаксический анализатор для понятия текст_со_скобками.
текст_со_скобками::= элемент или элемент текст_со_скобками
элемент::= A или B или (текст_со_скобками) или [текст_со_скобками] или
{ текст_со_скобками }

Описание функций.

bool isText(char* str) — главная задача данной функции это вызов рекурсивной функции Brackets и проверка значения, возвращаемого рекурсивной функцией. Если поданная функции строка удовлетворяет условию текста со скобками, то функция вернёт значение true, иначе false.

int Brackets(char* str, char OpenBrackets, int i, int RecDepth) — рекурсивная функция, принимает строку, символ (OpenBrackets) — открывающую скобку, для которой, по условию текста со скобками, в строке должна присутствовать парная закрывающая скобка, индекс с которого ведётся проверка в строке (i), счётчик глубины рекурсии (RecDepth).

bool PairScope(char* str, char OpenBracket, int i) — функция проверки парности скобок, принимает строку, символ, являющийся открывающей скобкой, индекс, по которому в строке должна находиться закрывающая скобка. Возвращает true, если удалось найти парную скобку, иначе false.

Описание алгоритма.

После вызова главной функции isText, принимающей строку для проверки функция проверяет строку на входящие в неё символы. При встрече элемента A или B функция пропустит его и перейдёт к проверке следующего, т.к строка с данными символами в любом количестве и месте по условию является текстом со скобками. При встрече открывающей скобки вызывается рекурсивная функция Brackets, которая при встрече элемента A или B также пропускает его, устанавливая в значении логической переменной AlphainScope истину. Если рекурсивная функция встречает элемент — открывающую скобку, то она вызывает себя же и возвращает значение в переменную i (index). При удачном завершении возвращает значение не равное -1, в этом случае устанавливает значение AlphainScope в истину, иначе возвращает -1. Brackets вернёт значение не равное -1 если последняя открывающая скобка является парной к закрывающей - рассматриваемой. Проверяет парность скобок функция PairScope. При выполнении условий — PairScope вернула истину и AlphainScope является истиной функция Brackets возвращает индекс следующего элемента за закрывающей скобкой. В конце, в случае строки, не являющейся текстом со скобками выводится символ и позиция символа в строке являющегося неверным.

Тестирование.

```
Введите текст со скобками: ((A[BAB])A)
RecFun('(')
    RecFun('(')
        RecFun('[')
        Найдена закрывающая скобка ']', удовлетворяющая условию, выход из рекурсивной функции
    Найдена закрывающая скобка ')', удовлетворяющая условию, выход из рекурсивной функции
Найдена закрывающая скобка ')', удовлетворяющая условию, выход из рекурсивной функции
Это текст со скобками
```

Ниже представлена таблица № 1, в которой представлены некоторые примеры работы программы, без отладочных выводов. Для удобства

максимальная глубина рекурсии в таблице выражена цифрами, а не отступами, как это сделано в программе.

Таблица №1 примеры работы программы.

Ввод	Вывод	Максимальная глубина рекурсии
([{ABAB}])	Это текст со скобками	3
()	Ошибка на символе номер 2 -) Это не текст со скобками	1
(AAAaBB(A))	Ошибка на символе номер 5 - a Это не текст со скобками	1
AAAA	Это текст со скобками	0
A(A(A))D	Ошибка на символе номер 8 - D Это не текст со скобками	2

Выводы.

В ходе лабораторной работы было изучены приёмы рекурсивного программирования, получены навыки программирования рекурсивных функций.

Исходный код.

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>

int Brackets(char* str, char OpenBracket, int i, int RecDepth);
bool isText(char* str);

/* функция проверки парности скобок, принимает
указатель на строку, индекс закрывающей скобки и открывающую скобку*/
bool PairScope(char* str, char OpenBracket, int i){
	if (OpenBracket == '(' && str[i] == ')') {
		return true;
```

```
		}
		else if (OpenBracket == '{' && str[i] == '}') {
			return true;
		}
		else if (OpenBracket == '[' && str[i] == ']') {
			return true;
		}
		else {
			return false;
		}
	}

	int main(int argc, char* argv[]) {
		char* str = (char*)calloc(200, sizeof(char));
		// если не указана папка с тестами, то пользователь вводи строку
		if (argc == 1){
			printf("Введите текст со скобками: ");
			fgets(str, 199, stdin);
				if (isText(str))
					printf("Это текст со скобками\n");
				else
					printf("Это не текст со скобками\n");
		}
		else {
			char* path = (char*)calloc(strlen(argv[1]) + 8, sizeof(char)); // строка для
открытия файла
			char* estr; // указатель для отслеживания конца файла
			strcat(path, "Tests/");
			strcat(path, argv[1]);
			FILE* fileTests = fopen(path, "r"); // открываем файл
			if (!fileTests){
				printf("Ошибка открытия файла\n");
				return 0;
			}
			else{
				while (1){
					estr = fgets(str, 199, fileTests); // считываем строку из файла
					// если указатель NULL то проверяем на ошибку или конец
файла
```

```
                                        if (estr == NULL){
                                                if (feof(fileTests) != 0){
                                                        printf("Конец файла\n");
                                                        break;
                                                }
                                                else {
                                                        printf("Ошибка чтения файла\n");
                                                        break;
                                                }
                                        }
                                        if (strlen(str) == 1){ // проверка на пустую строку
                                                printf("'Пустая строка'\n");
                                        }
                                        else{
                                                printf("%s\n", str);
                                        }
                                        if (isText(str)) // проверка на текст со скобками
                                                printf("Это текст со скобками  >> %s", str);
                                        else
                                                printf("Это не текст со скобками  >> %s", str);
                                }
                                free(path);
                        }
                }
                free(str);
                return 0;
}

// функция проверки, принимает указатель на строку
bool isText(char* str) {
        int i = 0; // индекс символа
        while (str[i] != '\n' || str[i] == '\0') {
                //пропуск букв тк их может сколько угодно идти подряд
                if (str[i] == 'A' || str[i] == 'B') {
                        i++;
                }
                else if (str[i] == '(' || str[i] == '[' || str[i] == '{') {
                        i++;
```

```
                                        // находим скобку и вызываем рекурсивную функцию, возвращаем
индекс с которого продолжим
                                        i = Brackets(str, str[i - 1], i, 0);
                                        if (i == -1) {
                                                return false;
                                        }
                                }
                                else {
                        printf("Ошибка на символе номер %d - %c \n", i + 1, str[i]);
                                        return false;
                        }
                        }
                        // возвращаем истину, тк дошли до конца без ошибок и проверяем на не пустоту
строки.
                        return true && i;
                }

                /* рекурсивная функция принимает указатель на строку
                символ который является открывающей скобкой индекс текущего символа, глубину
рекурсии */
                int Brackets(char* str, char OpenBracket, int i, int RecDepth) {
                        bool AlphainScope = false; // наличии буквы в скобках
                        printf("%*.sRecFun('%c')\n", 4*RecDepth, " ", str[i - 1]); // отладочный вывод
                        while (str[i] != '\n'|| str[i] == '\0') {
                                if (str[i] == 'A' || str[i] == 'B') {
                                        AlphainScope = true;
                                        i++;
                                }
                                // аналогично работе в функции isText
                                else if (str[i] == '(' || str[i] == '[' || str[i] == '{') {
                                        i++;
                                        // возвращаем индекс с которого будем продолжать
                                        i = Brackets(str, str[i - 1], i, RecDepth + 1);
                                        if (i != -1) {
                                                AlphainScope = true;
                                        }
                                        else {
                                                return -1;
                                        }
```

```
                        }
                        // проверка на парсность скобок и вложенность в них символа
                        else if (PairScope(str, OpenBracket, i) && AlphainScope) {
                                i++;
                                printf("%*.s", 4*RecDepth, " "); // отладочные выводы
                                printf("Найдена закрывающая скобка '%c', удовлетворяющая
условию, выход из рекурсивной функции\n", str[i - 1]);
                                return i;
                        }
                        else {
                                printf("Ошибка на символе номер %d - %c \n", i + 1, str[i]);
                                break;
                        }
                }
                return -1;
        }
```